(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 April 2001 (12.04.2001)

PCT

(10) International Publication Number
WO 01/25894 A1

(51) International Patent Classification[7]: G06F 3/14

(21) International Application Number: PCT/US00/27640

(22) International Filing Date: 5 October 2000 (05.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/157,727	5 October 1999 (05.10.1999)	US
60/157,728	5 October 1999 (05.10.1999)	US
60/157,729	5 October 1999 (05.10.1999)	US
60/157,833	5 October 1999 (05.10.1999)	US
60/157,834	5 October 1999 (05.10.1999)	US

(71) Applicant *(for all designated States except US)*: EJASENT INC. [US/US]; 2490 Charleston Road, Mountain View, CA 94043 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: HIPP, Burton, A. [US/US]; 4117 E. Haack Ct., Elk Grove, CA 95758 (US). BHARADHWAJ, Rajeev [US/US]; 1405 Redwood Drive, Los Altos, CA 94024 (US).

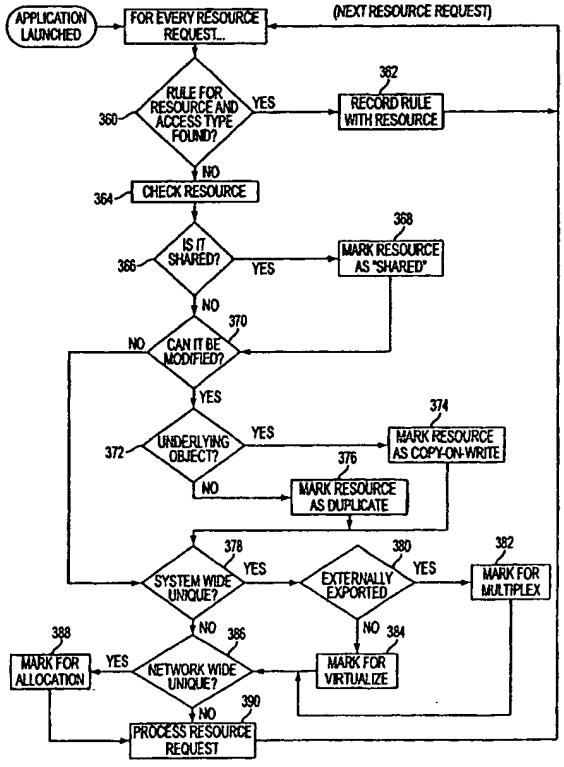
(74) Agents: ASHBY, David, C. et al.; Flehr Hohbach Test Albritton & Herbert LLP, 4 Embarcadero Center, Suite 3400, San Francisco, CA 94111-4187 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

[Continued on next page]

(54) Title: SNAPSHOT VIRTUAL-TEMPLATING



(57) Abstract: The present invention saves all process, state, memory, and dependencies related to a software application to a snapshot image. Interprocess communication (IPC) mechanisms such as shared memory (366, 368) and semaphores must be preserved in the snapshot image. IPC mechanisms include any resource that is shared between two process or any communication mechanism or channel that allow two processes to communicate or interoperate is a form of IPC. At snapshop time, state is saved by querying the operating system kernel, the application snapshot/restore framework components, and the process management subsystem that allows applications to retrieve internal process-specific information not available through existing system calls.

WO 01/25894 A1

patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *With international search report.*

SNAPSHOT VIRTUAL-TEMPLATING

REFERENCE TO RELATED APPLICATIONS

The present application claims priority to and incorporates the following applications by reference: DYNAMIC SYMBOLIC LINK RESOLUTION, Prov. No. 60/157,728, filed on October 5, 1999; SNAPSHOT VIRTUAL TEMPLATING, Prov. No. 60/157,728, filed on October 5, 1999; SNAPSHOT RESTORE OF APPLICATION CHAINS AND APPLICATIONS, Prov. No. 60/157,833, filed on October 5, 1999; VIRTUAL RESOURCE-ID MAPPING, Prov. No. 60/157,727, filed on October 5, 1999; and VIRTUAL PORT MULTIPLEXING, Prov. No. 60/157,834, filed on October 5, 1999.

FIELD OF THE INVENTION:

The present invention relates broadly to client server computer architectures. Specifically, the present invention relates to creating virtual application templates for the purpose of propagating a single application snapshot into multiple, distinct images.

BACKGROUND

Global computer networks such as the Internet have allowed electronic commerce ("e-commerce") to flourish to a point where a large number of customers purchase goods and services over websites operated by online merchants. Because the Internet provides an effective medium to reach this large customer base, online merchants who are new to the e-commerce marketplace are often flooded with high customer traffic from the moment their websites are rolled out. In order to effectively

serve customers, online merchants are charged with the same responsibility as conventional merchants: they must provide quality service to customers in a timely manner. Often, insufficient computing resources are the cause of a processing bottleneck that results in customer frustration and loss of sales. This phenomena has resulted in the need for a new utility: leasable on-demand computing infrastructure. Previous attempts at providing computing resources have entailed leasing large blocks of storage and processing power. However, for a new online merchant having no baseline from which to judge customer traffic upon rollout, this approach is inefficient. Either too much computing resources are leased, depriving a start up merchant of financial resources that are needed elsewhere in the operation, or not enough resources are leased, and a bottleneck occurs.

To make an on-demand computer infrastructure possible, computer applications must be ported across computer networks to different processing locations. However, this approach is costly in terms of overhead for the applications to be moved across the network must be saved, shut down, stored, ported and then restored and re-initialized with the previously running data. The overhead is prohibitive and negates any performance improvements realized by transferring the application to another computer. Thus, there remains a heartfelt need for a system and method for effecting a transfer of applications across computer networks without incurring costly processing overhead.

SUMMARY

The present invention solves the problems described above by creating virtual application templates for the purpose of propagating a single application snapshot into multiple, distinct images. Snapshot virtual templates allow multiple application instances to use the same fixed resource identifier by making the resource identifier virtual, privatizing it, and dynamically mapping it to a unique system resource identifier. When a snapshot is cloned from a virtual template, the common or shared data is used exactly as is, whereas the non-sharable data is either copied-on-write, multiplexed, virtualized, or customized-on-duplication. The present invention greatly reduces the required administrative setup per application instance. Snapshot virtual templating works by noting access to modified resources, fixed system IDs/keys and unique process-related identifies and automatically inserting a level of abstraction

between these resources and the application. The resources contained in a snapshot virtual template can be dynamically redirected at restore time. Access to memory and storage is managed in a copy-on-write fashion. System resource handles are managed in a virtualize-on-allocate fashion or by a multiplex-on-access mechanism. Process-unique resources are managed in a redirect-on-duplicate fashion. Rules may be defined through an application configurator that allows some degree of control over the creation of non-sharable data.

The snapshot virtual template is constructed by dividing the snapshot image into sharable and non-sharable data. Knowledge of which system resources can be shared is encoded in the application snapshot/restore framework.

These and many other attendant advantages of the present invention will be understood upon reading the following detailed description in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a high level block diagram illustrating the various components of a computer network used in connection with the present invention;

FIG. 2 is a high level block diagram illustrating the various components of a computer used in connection with the present invention;

FIG. 3 illustrates how application state is tracked using library and operating system kernel interposition;

FIG. 4 illustrates the capture of an application's run-time state;

FIG. 5 is a flow chart illustrating the logical sequence of steps executed to create a snapshot image of an application instance;

FIG. 6 is a flow chart illustrating the logical sequence of steps executed to restore an application instance from a snapshot image.;

FIG. 7 is an illustration of the format of a snapshot virtual template;

FIG. 8 is a flowchart illustrating the logical sequence of steps executed to create a snapshot virtual template;

FIG. 9 is a flowchart illustrating the logical sequence of steps executed to clone a snapshot virtual template;

FIG. 10 illustrates the registration of an application using virtual resource identifiers;

FIG. 11 illustrates the allocation of a virtual resource;

FIG. 12 illustrates the translation of a virtual resource to a system resource;

FIG. 13 illustrates the translation of a system resource to a virtual resource;

FIG. 14 is a flowchart illustrating the logical sequence of steps executed to create a virtual translation table; and

FIG. 15 is a flowchart illustrating the logical sequence of steps executed to translate a virtual resource.

DETAILED DESCRIPTION

A. Snapshot Restore

FIG. 1 illustrates in high level block diagram form the overall structure of the present invention as used in connection with a global computer network 100 such as the Internet. Remote users 102-1 and 102-2 can connect through the computer network 100 to a private network of computers 106 protected by firewall 104. Computer network 106 is a network comprising computers 150-1, 150-2, through 150-n, where n is the total number of computers in network 106. Computers 150 are used to run various applications, as well as host web sites for access by remote users 102. The present invention is implemented on computer network 106 in the form of virtual environments 110-1 and 110-2. While only two virtual environments are illustrated, it is to be understood that any number of virtual environments may be utilized in connection with the present invention.

FIG. 2 illustrates in high level block diagram form a computer that may be utilized in connection with the present invention. Computer 150 incorporates a processor 152 utilizing a central processing unit (CPU) and supporting integrated circuitry. Memory 154 may include RAM and NVRAM such as flash memory, to facilitate storage of software modules executed by processor 152, such as application snapshot/restore framework 200. Also included in computer 150 are keyboard 158, pointing device 160, and monitor 162, which allow a user to interact with computer 150 during execution of software programs. Mass storage devices such as disk drive 164 and CD ROM 166 may also be in computer 150 to provide storage for computer programs and associated files. Computer 150 may communicate with other computers via modem 168 and telephone line 170 to allow the computer 150 to be operated remotely, or utilize files stored at different locations. Other media may also be used in place of modem 168 and telephone line 170, such as a direct connection or high

speed data line. The components described above may be operatively connected by a communications bus 172.

FIG. 3 shows how application states are tracked via library and kernel interposition. The application snapshot/restore framework 200 is a software module that processes transactions between the operating system 206 and the applications 208. Requests for system resources or changes to process state are routed internally and the application snapshot/restore framework 200 tracks these events in anticipation of a snapshot. The application snapshot/restore framework 200 is transparent to running (and snapshotted) applications. From an application's perspective, the application is always running. An application snapshot may consist of multiple processes and multiple threads and includes shared resources in use by a process, such as shared memory or semaphores. A process may be snapshotted & restored more than once. The computer on which a process is restored on must be identically configured and have an identical environment (hardware, software, and files) that matches the environment of the computer where the process was snapshotted. All processes that are snapshotted together in the form of an application chain share the same application ID ("AID"). As used herein, an application chain is the logical grouping of a set of applications and processes that communicate with each other and share resources to provide a common function.

The virtual environment 204 is a layer that surrounds application(s) 208 and resides between the application and the operating system 206. Resource handles are abstracted to present a consistent view to the application although the actual system resource handles may change as an application is snapshot/restored more than once. The virtual environment also allows multiple applications to compete for the same resources where exclusion would normally prohibit such behavior to allow multiple snapshots to coexist without reconfiguration. Preload library 214 is an application library that interposes upon an application for the express purpose of intercepting and handling library called and system calls. Once the library has been preloaded it is attached to the process' address space. Preload library 214 interposes between application 208 and operating system 206. It is distinguished from kernel interposition in that it operates in "user mode" (i.e., non-kernel and non-privileged mode). Application 208 can make application programming interface (API) calls that modify the state of the application. These calls are made from the application 208 to the

operating system API interfaces 210 via the application snapshot restore framework 200 or the preload library 214. The preload library can save the state of various resources by intercepting API interface calls and then saves the state at a pre-arranged memory location. When the process' memory is saved as part of the snapshot/restore mechanism, this state is saved since it resides in memory. The state as it is modified is saved to non-volatile storage (i.e. a file on disk). The preload library notify the snapshot/restore framework through one of its private interface.

FIG. 4 illustrates the capture of an application's run time state. The OS API interfaces 210 are standard programming interfaces defined by international standards organizations such as XOPEN. The open() system call which allows an application to open a file for reading is an example of an API interface. The process management system 216 is a component of the operating system 206 that allows one process to examine or alter the state of another process. The interfaces that are provided by this component are usually not standardized interfaces (not part of a recognized standard API) and are OS-implementation dependent. However, such interfaces usually allow access to more state than standardized API interfaces. The run-time information captured from a process is used by the snapshot driver 218.

An application needs to be snapshotted if it is idle and is not currently in use or there are higher priority requests that require the application be scheduled out and preempted in favor of another application. A snapshot request is initiated by an application scheduler that determines when an application needs to be restored on-demand and when the application is no longer needed (can be snapshotted to free up resources). The application scheduler does this based on web traffic, server load, request response time, and a number of other factors. An application needs to be restored if there is an incoming request (i.e. a web browser request) and the application required to handle that request (ie a particular web site) is not currently running. Alternatively, an application needs to be restored if there is an incoming request (i.e. a web browser request) and the application required to handle that request (ie a particular web site) is currently overloaded, so another instance of that application is restored to handle that request.

FIG. 5 illustrates the logical sequence of steps executed by Snapshot driver 218 to make a snapshot image of a process. Beginning at step 250, an snapshot image of a runnable application is requested. The AID is looked up (decision step 252) in a table

in memory 154 containing a list of every AID present on computer 150. If the AID is not found control returns at step 254. However, if the AID is found, control continues to decision step 256 where the snapshot/restore framework 200 searches for a process belonging to the application having the matched AID. If a process is found, control continues to step 258, where the process is suspended. For a process to be snapshotted, it must be completely suspended with no activity present and no ambiguous state (i.e., in a transitory state). Since a process may be represented by asynchronous threads of activity in the operating system that are not formally part of the process state, any activity in the operating system that is executing on behalf of the process must be stopped (i.e. disk I/O activity). In other words, there may be moments where temporarily a process cannot be snapshotted. This is a finite and short period of time, but it can occur. If the state is consistent and the threads are quiesced (decision step 260), control loops to step 256 and the remaining processes belonging to the application are located and suspended. However, if a process is located that does not have a consistent state or a thread is not quiesced, suspended processes are resumed and the snapshot cannot be completed.

Once all related processes are suspended, for each state of each suspended process, the state is checked to see if it is virtualized (step 264). A virtualized state is any process state that reflects a virtualized resource. If the state is virtualized, it is retrieved at step 266 ; otherwise the non-virtualized state is retrieved at step 268 State retrieval is performed as described above by the snapshot driver 218 querying the application snapshot/restore framework 200, operating system API interfaces 210, and process management subsystem 216. If the state has changed since the last snapshot (step 270), the new state is recorded. Control then loops to step 264 and executes through the above sequence of steps until all states of all processes are checked. Once completed, control proceeds to step 278, the registered global state, such as semaphores, is removed. Registered global state is state that is not specifically associated with any one process (ie private state). Global state is usually exported (accessible) to all processes and its state can be modified (shared) by all processes. Control proceeds to step 280, where the process is terminated. If there are remaining processes (step 282), these are also terminated. This sequence of steps is concluded to create a snapshot image which is stored as a file and made available for transmission

to another computer within public computer network 100 or private computer network 106.

FIG. 6 illustrates the sequence of steps executed by the restore driver 220 to restore a snapshot image. The snapshot image is accessed via a shared storage mechanism and a restore call is made at step 300. The restore driver 220 looks up the AID for the snapshot image and (decision step 302) if not found control returns and the snapshot image cannot be restored. However, if the AID is found, control continues to decision step 304 where, if the snapshotted image matching the AID is located, the global/shared state for each process associated with the snapshot are found. Control then continues to step 308, where remaining global or shared state for the processes are recreated. Since global and shared state is not associated with a single process and may be referenced by multiple processes, it is created first. Recreating the state entails creating a global resource that is functionally identical to the resource at the time of the snapshot. For example if during a snapshot, a semaphore with id 5193 is found with a value of 7, then to recreate the state at restore time a new semaphore must be created having the exact same ID as before (ie 5193) and it also must have the same state (ie value 7) as before. Then, for each image, a process is created that inherits the global/shared state restored in step 308, and each created process is isolated to prevent inter-process state changes. When a process is being restored, process state is being registered with the kernel, inter-process mechanisms are being restored and reconnected and I/O buffers in the kernel may be being restored. Some of these actions in one process may have the unintended side effect of disturbing another process that is also being restored. For example if an I/O buffer that is in the operating system as a result of a $process_x$ performing a write to a socket connection, then $process_y$ could unintentionally be delivered an asynchronous signal that notifies it of I/O being present (for reading) prior to the process being fully restored. At step 314, for each type of state within the processes, the process-private resources are recreated to their state at the time the snapshot image was taken. If the state is virtualized (decision step 316), the system state is bound to a virtual definition. As part of the restore an extra step must be done to create a virtual mapping. This is done by taking the system resource that was created in step 314 and binding it to the virtual definition that was saved during the snapshot in step 266. This allows the application to see a

consistent view of resources, since it cannot be guaranteed that at restore time the exact same system resource will be available. If the state is shared with another process, such as via a pipe (decision state 320), the shared state is reconnected with the other process at step 322. If there are more states (decision step 324) steps 314 through 322 are repeated. Once steps 314 through 322 have been executed for all states, control continues to step 326, where the process is placed in synchronized wait. If there are remaining images in the snapshot image (decision step 328), steps 310 through 326 are repeated. Once all images have been processed, control continues to step 330, where traces and states induced during restore of the process are removed, and a synchronized resume of all processes occurs at step 332.

Once steps 300 through 332 have executed without error on whatever computer the restore driver 220 was executed, the restored application can continue to run without interruption. Thus, the present invention avoids the overhead and delay of shutting down an application, storing data to a separate file, moving both the application and data file elsewhere, and restarting the program.

B. Snapshot Virtual Templating

In another aspect, the present invention provides a system, method, and computer program product for creating snapshot virtual application templates for the purpose of propagating a single application snapshot into multiple distinct instances. Snapshot virtual templates allow multiple application instances to use the same fixed resource ID ("RID") by making the resource ID virtual, privatizing the virtual RID, and dynamically mapping it to a unique system resource ID. A RID is the identifier assigned to represent a specific system resource and acts as a handle when referencing that system resource. Anonymous resources are resources that are read-only or functionally isolated from other applications. Anonymous resources are also shareable resources. An anonymous resource is a non-fixed resource allocated by the operating system and identified by a per-process handle. These are functionally-isolated since the operating system allocates it anonymously and one is as food as another. Examples of this are non-fixed TCP ports or file descriptors. A resource is said to be network-wide unique if there can only be one instance of that resource with its corresponding identifier on computer network or subnetwork. An example of this is an network IP address (i.e. 10.1.1.1). Snapshot virtual templates allow snapshots to be

described in a manner that separates shareable data from non-sharable data. Data is loosely defined to mean any system resource (memory, files, sockets, handles, etc.). When a snapshot is cloned from a virtual template, the common or shared data is used exactly as is, whereas the non-sharable data is either copied-on-write, multiplexed, virtualized, or customized-on-duplication. The present invention greatly reduces the required administrative setup per application instance. Snapshot virtual templating works by noting access to modified resources, fixed system IDs/keys and unique process-related identifies and automatically inserting a level of abstraction between these resources and the application. The resources contained in a snapshot virtual template can be dynamically redirected at restore time. Access to memory and storage is managed in a copy-on-write fashion. System resource handles are managed in a virtualize-on-allocate fashion or by a multiplex-on-access mechanism. Process-unique resources are managed in a redirect-on-duplicate fashion. Rules may be defined through an application configurator that allows some degree of control over the creation of non-sharable data.

The application configurator is a software component that resides in the application domain and communicates configuration information about the application on its behalf such as the DSL specifications. Since this component operates without assistance from the application, it may exist in the form of an application library, or may be placed in the applications environment (via inheritance at execution time), ir it can be implemented as a server process that proxies application information to the operating system as necessary.

A resource duplicator is a software component that fields requests for non-shareable resources and duplicates or virtualizes resources so that applications receive their own private copies and can co-exist transparently with multiple instances of the same application forged from the same virtual template. The resource duplicator also processes duplication rules fed by the application configurator or application snapshot/restore framework 200.

As used herein, non-sharable data refers to any resource that is modified and globally visible to other application instances is non-sharable (i.e. files). Process-related identifiers that are system-wide unique are also non-shareable since conflicts will arise if two instances use the same identifier at the same time (uniqueness is no longer preserved). References to unique resources by fixed handles (i.e. fixed TCP

port numbers or IPC keys) are also not shareable. Memory pages that are specific to an application instance (i.e. the stack) are another example of a non-shareable resource. For illustrative purposes, examples of non-sharable data include application config files that must be different per application instance as well as modified application data files if the application is not written to run multiple copies simultaneously. Other examples include stack memory segments or heap segments may also be non-sharable data, shared memory keys that are a fixed value, usage of fixed well-known (to the application) TCP port numbers, and process identifiers (two distinct processes cannot share the same PID).

The snapshot virtual template is constructed automatically by dividing a snapshot process into shareable and non-shareable data. The knowledge of which system resources can be shared is encoded in the snapshot virtual templating framework itself. If an application has non-shareable internal resources (as opposed to system resources), it may not be possible to construct a virtual-template for that application.

Snapshot virtual templates are node-independent as well as application-instance dependent. Snapshot virtual templates cannot be created for applications that use non-shareable physical devices. Snapshot virtual templates must save references to non-shareable resources in their pre- customized form, rather than their evaluated form. All access by an application to non-shareable resources must be via the operating system. Internal or implicit dependencies by the application itself cannot be virtually-templated. A snapshot virtual template may be created from an application instance that was originally forged from a different virtual template.

Snapshot virtual templating is an alternate method of creating an application instance. The snapshot restore method described above requires creating unique instances of an application to create unique "snapshots" of that application. Virtual templating allows the creation of a generic application instance from which unique instances may be spawned. Every unique instance that is created from the original virtual template starts out as an exact copy (referred to herein as "clone") but has been personalized just enough to make it a fully-functioning independent copy. Differences between copies may be due to the way resources are named or identified.

FIG. 7 illustrates in block diagram form the contents of a snapshot virtual template. The main components are resource name size, resource descriptor size,

resource type, resource name, and resource data. Resource data includes many different types of information, as illustrated.

FIG. 8 describes the sequence of steps executed by the application snapshot/restore framework 200 to create a snapshot virtual template. As the application runs, every request for a new operating system resource (file, memory, semaphore, etc.) is checked for an existing rule. When the application is started under the virtual templating framework, a set of rules may be supplied at that time. The rule will state the type of resource, the type of access (i.e., create, read, write, destroy, etc), and the action to be taken. If a rule is found (decision step 360), the rule is saved as part of the process state and recorded with the resource as auxiliary state at step 362. Rules may be added to the template that control the creation of application-instance specific resources. For example, environment variables or pathnames that incorporate an AID to differentiate and customize a particular resource among multiple instances. The following syntax created for illustration purposes:

Define	<APPL-ID> as PROPERTY application-ID
REDIR PATH	"/user/app/config" to "/usr/app/<APPL-ID>/config"
SET ENV	"HOME" = "/usr/app/<APPL-ID>"

If rules are created, they should also be specified via the application configurator. If no rule is found, the resource is checked using a standard set of criteria that determine whether the resource needs to be abstracted or virtualized in order to be cloned at step 364. The criteria is again checked at steps 366, 370, 372, 378, 380 and 386. In most cases, no action is taken. Resources are simply classified into their correct types so that when an instance is cloned the correct action can be taken. If the resource is shared, i.e. shared memory (decision step 366), the resource is marked as shared (step 368) so that during the subsequent snapshot all references to the shared object will be noted. If the resource can be modified (decision step 370), it must be isolated from the original during cloning so that the original remains untouched. If the resource is a large object and has a notion of an underlying object, such as i.e. mapped memory (decision step 372), it is marked for copy-on-write (step 374). Otherwise, the entire resource must be duplicated and marked accordingly (step 376). A resource is said to be systemwide unique if the identifier used to represent

that resource cannot represent more than one instance of that resource at a single point in time on the same node or computer. If the resource is systemwide unique (decision step 378), and is exported as an external interface, as is the case when another client application that is not running on the platform has a dependency on the resource, such as a TCP port number (decision step 380), it is marked to be multiplexed (step 382). Multiplexing allows multiple independent connections to transparently co-exist over the same transport using only a single identifiable resource. If it isn't externally exported, the resource is marked for virtualize at step 384. Continuing to decision step 386, if the resource is network-unique, it is marked for allocation at step 388. Control proceeds to step 390, where the resource request is processed. Steps 360 through 390 are repeated for every resource request occurring during application execution.

FIG. 9 illustrates the sequence of steps executed to perform cloning or replication of a process from a snapshot virtual template. This sequence of steps can be performed by a replication program that creates a new snapshot image from an existing template, or by the restore driver 220. When an application instance is restored from a snapshot that is a virtual template, a new instance is automatically cloned from the template using the rules that were gathered during the creation of the template. For every resource included in the snapshot virtual template, rules for the resource and access type are looked up. Any resource that requires special handling as part of the templating effort has the rule described inside the snapshot template as part of the auxiliary state associated with the resource. If no rule is found (decision step 400), the resource is recreated using the existing saved information in the snapshot (step 402). Otherwise, if a resource is marked for duplicate (decision step 404), then a copy of the original resource is made at step 406. If a resource is marked for copy-on-write (decision step 408), then at step 410 a reference to the original underlying object (in the original template) is kept, and any modifications to the original force a copy-on-write action so that the modifications are kept in an application-instance private location and the two form a composite view that is visible to the application instance.

If a resource is marked for virtualization (decision step 412), the original resource is allocated or duplicated in blank form at step 414. At step 416, the resource is mapped dynamically to the new resource at run-time by binding the system resource to the saved resource in the snapshot image. If a resource is marked for multiplex (decision step 418), the original resource is duplicated and then spliced

among other application instances that share it (step 420). If the resource is a network unique resource (decision step 422), a unique resource must be allocated (step 424) by communicating with another component of the network, i.e. network map or registry, that assigns a resource to this instance. Then this new resource is bound to the fixed resource that was saved in the virtual template (step 426), in a manner similar to virtualization.

C. Virtual Resource ID Mapping

The present invention provides virtual mapping of system resource identifiers in use by a software application for the purpose of making the running state of an application node independent. By adding a layer of indirection between the application and the resource, new system resources are reallocated and then can be mapped to the application's existing resource requirements while it is running, without the application detecting a failure or change in resource handles.

This layer of indirection makes the application's system RID transparent to the application. RID's are usually numeric in form, but can also be alphanumeric. RID's are unique to a machine, and can be reused once all claims to a specific RID have been given up. Some examples of RID's include process ID's, shared memory ID's, and semaphore ID's. Only the virtual RID is visible to the application. Conversely, the virtual RID is transparent to the OS, and only the system RID is visible to the OS. Every application has a unique identifier that distinguishes it from every other running application. There exists a one to one mapping between the AID: resource type: virtual RID combination and the node ID: system RID. Virtual RID's are only required to be unique within their respective applications, along with their corresponding system RID's may be shared among multiple programs and processes that have the same AID. System RID's that have been virtualized are accessed through their virtual ID's to ensure consistent states.

AID's are farm-wide unique resources and are allocated atomically by the AID generator. Because in the present invention applications aren't uniquely bound to specific names, process ID's, machine hostnames or points in time, the AID is the sole, definitive reference to a running application and its processes. Typically, an AID is defined in reference to a logical task that the application is performing or by the logical user that is running the application.

Virtual resource mapping comprises several basic steps: application registration, allocation of the RID, and resolution of the RID. During registration of the application, the AID is derived if preallocated or the application existed previously, or it may be allocated dynamically by an AID generator. The AID is then made known for later use. Allocation of a RID happens when an application requests access to a system resource (new or existing) and the OS returns a handle to a resource in the form of a RID. The virtual resource layer intercepts the system returned RID, allocates a virtual counterpart by calling the resource specific resource allocator, establishes mapping between the two, and returns a new virtual RID to the application.

Resolution of a RID may occur in two different directions. A RID may be passed from the application to the OS, in which case the RID is mapped from virtual ID to system ID. Conversely, the RID may be passed from the OS to the application, in which case the transition is from system ID to virtual ID. Requests for translation are passed from the framework to the virtual RID translation unit and the corresponding mapping is returned once it has been fetched from the appropriate translation table. Multiple translation tables may exist if there are multiple resource types.

FIG. 10 illustrates the steps executed to register an application. The AppShot harness 500 exists to aid in the creation of the appropriate runtime environment for the application prior to launching the application. The appshot harness 500 initializes the runtime environment for an application by first priming its own environment with the appropriate settings and then launching the application which inherits all these settings. The appshot harness 500 is provided because the application cannot be recompiled or rewritten to initialize its own environment. Some of the settings that are established by the appshot harness 500 include the AID, assigned process ID range, DSL specifications, application virtual ID's, and snapshot templating rules. The DSL specifications are registered as part of the environment. A process is an in-memory instantiation of a software program. $Process_x$ is the in-memory image of the AppShot harness 500 and $process_y$ is the in-memory image of the application. At step 510, the appshot harness 500 registers an AID a_i, such as "dbserver." with the application snapshot/restore framework 200 within the OS kernel 206. The application snapshot/restore framework 200 then creates virtual translation tables 502 for the AID at step 512. Virtual translation tables 502 are data units that contain translation

information related to RID's, such as AID's or process ID's, virtual RID's, and system RID's. Separate tables can be implemented per resource type or a table can be shared if a unique resource type is stored per table entry. A translation unit maps the system RID's to the virtual RID's by storing and fetching translation information in the appropriate translation table. Once the virtual translation tables 502 are created, $Process_x$ is linked to the AID a_i at step 514. At this point, $process_y$ is created when the appshot harness 500 launches the application at step 516. $Process_y$ then inherits $process_x$'s link to a_i and its tables at step 518.

FIG. 11 shows the allocation of a virtual resource such as a semaphore in accordance with the present invention. At step 520, an application requests that a semaphore resource is allocated for its $process_y$. In response (step 522), the application snapshot/restore framework 200 looks up the AID in memory 154, and returns AID a_i. At step 524, in response to a request from the application snapshot/restore framework 200, the system semaphore pool returns semaphore s_i. At step 526, the application snapshot/restore framework 200 scans the virtual resource translation table 502 for an available slot and allocates the virtual semaphore. At step 528, the application snapshot/restore framework 200 inserts the translation $s_i = a_i:v_3$ and the virtual resource translation table 502 now contains the mapping. At step 530, the virtual semaphore v_3 is returned to the application.

FIG. 12 illustrations translation of a virtual resource to a system resource in accordance with the present invention. At step 532, the application calls the semaphore interface and supplies the virtual RID v_3 to the application snapshot/restore framework 200. At step 534, the application snapshot/restore framework 200 looks up the AID for the calling application and returns a_i. At step 536, the application snapshot/restore framework 200 then looks up the translation for $a_i:v_3$ in the virtual resource translation table 502, which returns s_i at step 538. At step 540, the OS semaphore implementation is achieved when the application snapshot/restore framework 200 forwards the application's request by substituting s_i for v_3.

FIG. 13 illustrates translation of a system resource to a virtual resource. Beginning at step 542, the application calls the semaphore interface and expects the RID as a result. At step 544, the application snapshot/restore framework 200 looks up the AID for the calling application and returns a_i. At step 546, the application

snapshot/restore framework 200 forwards the application request to the OS semaphore implementation, which returns the system semaphore s_i at step5 48. At step 550, the application snapshot/restore framework 200 then looks up the translation for $a_i:s_1$ in the virtual resource translation table 502, which returns v_3 at step 552. At step 554, the application snapshot/restore framework 200 returns the virtual RID v_3 to the calling application.

FIG. 14 illustrates the logical sequence of steps executed to create the virtual translation table 502. Beginning at step 556, and attempt is made to register AID a_i. If the AID hasn't already been registered (decision step 558), a virtual resource translation table space for a_i is created in table 502 (step 560). Translation tables are then added for each type of resource associated with the application at step 562. At step 564, the $process_x$ is linked to a_i. At step 566, the $process_y$ is created and the application is launched. Steps 568 and 570 show the parallel paths of execution. The flow of control continues on to step 568 and halts shortly thereafter. The new flow of execution continues on from 566 to 570, where the process inherits context from the process at step 568.

FIG. 15 illustrates in greater detail the sequence of steps executed to translate a virtual resource. For illustrative purposes, the resource in this example is a semaphore. Beginning at decision step 580, if an AID is found upon lookup, and the interface uses the RID as a parameter (decision step 582), the application snapshot/restore framework 200 performs a lookup of the translation for $a_i:v_i$at step 584. If a system resource s_1 is found, the system RID is substituted for the virtual RID at step 586 and passed to the semaphore interface of the OS 206 (step 588). If the semaphore was not allocated by the semaphore interface (decision step 590), and the interface returns a semaphore (decision step 592) control proceeds to step 594 where a reverse lookup for the translation of the AID with a system RID is performed. The returned virtual RID is then substituted for the virtual ID at step 596. Returning to decision step 590, if a semaphore was allocated by the semaphore interface, control proceeds to step 598 where the virtual semaphore is allocated and a translation for $v_2=$ $a_1:s_2$ is inserted into the translation table 502 at step 600. V_2 is then substituted for s_2 at step 602.

In another aspect, the present invention provides communication between at least two applications through virtual port multiplexing. The communication is

achieved by accepting a connection from a second application on a first port and allocating a second port to receive the communication from the second application. Once the second port has been allocated the second port translation is recorded. The communication is sent to the first port from the second application and received on the second port. The communication is then delivered to a first application from the second port. In one embodiment the first application requests the communication from the first port and the first port is translated to determine the second port such that the communication is delivered to the first application in the step of delivering the communication to the first application.

In one embodiment, the communication is received on the first port following the step of sending the data to the first port, the first port is translated to determine the second port prior to the step of receiving the communication on the second port, and the step of receiving the communication on the second port includes queuing the communication on the second port from the first port.

In one embodiment, the second application requests to connect with the first port prior to the step of accepting the connection. Once the second port is allocated, the second port is negotiated including negotiating the second port between a first and second virtual port multiplexer. Further, the second application is connected with the second port following the step of allocating the second port. The step of recording the translation including, first, recording the translation of the second port in association with the first application, and second, recording the translation of the second port in association with the second application.

The present invention also provides for a dynamic symbolic link (DSL) and the resolution of that DSL. The pathname of a first application is renamed to a target pathname, a variable within the target pathname, the first pathname is defined as a symbolic link and the symbolic link is associated with a virtual pathname. The method and apparatus further defines a specification is further defined that is associated with the virtual pathname including associating the variable with the virtual pathname. In associating the symbolic link with the virtual pathname, a declaration is defined within the virtual pathname.

Having disclosed exemplary embodiments and the best mode, modifications and variations may be made to the disclosed embodiments while remaining within the scope of the present invention as defined by the following claims.

CLAIMS

What is claimed is:

1. In a computer system, a method for propagating a single snapshot image of a running software application into multiple instances, wherein the snapshot image includes at least one process, state information associated with the process and an identifier associated with the application, said running application restored on a similarly configured computer having an operating system and having an environment similar to the environment of the computer where the process was snapshot image was produced, comprising the steps of:

(a) dividing the snapshot image into shareable data and non-shareable data;

(b) producing at least one additional instance of the snapshot image upon restoration of the snapshot image;

(c) providing the sharable data to said additional instance; and

(d) controlling access by the additional instance to non-shareable data.

2. The method of claim 1, wherein said non-shareable data is copied on a write operation.

3. The method of claim 1, wherein said non-shareable data is multiplexed.

4. The method of claim 1, wherein said non-shareable data is virtualized.

5. The method of claim 1, wherein said additional instance is altered to provide customization of the application.

6. The method of claim 5, wherein references to non-shareable resources are saved in original form.

7. A computer program product for propagating a single snapshot image of a running software application into multiple instances, wherein the snapshot image includes at least one process, state information associated with the process and an identifier associated with the application, said running application restored on a

similarly configured computer having an operating system and having an environment similar to the environment of the computer where the process was snapshot image was produced, comprising the steps of:

(a) dividing the snapshot image into shareable data and non-shareable data;

(b) producing at least one additional instance of the snapshot image upon restoration of the snapshot image; and

(c) providing private copies of non-sharable data to said additional instance, wherein access to non-shareable resources is controlled by the operating system.

8. A system for propagating a single snapshot image of a running software application into multiple instances, wherein the snapshot image includes at least one process, state information associated with the process and an identifier associated with the application, said running application restored on a similarly configured computer having an operating system and having an environment similar to the environment of the computer where the process was snapshot image was produced, comprising

(a) a first software module that processes transactions between the operating system and application and tracks requests for system resources or changes to process state in anticipation of a snapshot;

(b) a storage format for organizing an application snapshot image into shareable and non-shareable data, said storage format manipulated by said software module; and

(c) a second module that fields requests for non-shareable resources, duplicates said non-shareable resources and provides said duplicated non-shareable resources to application instances produced from said storage format.

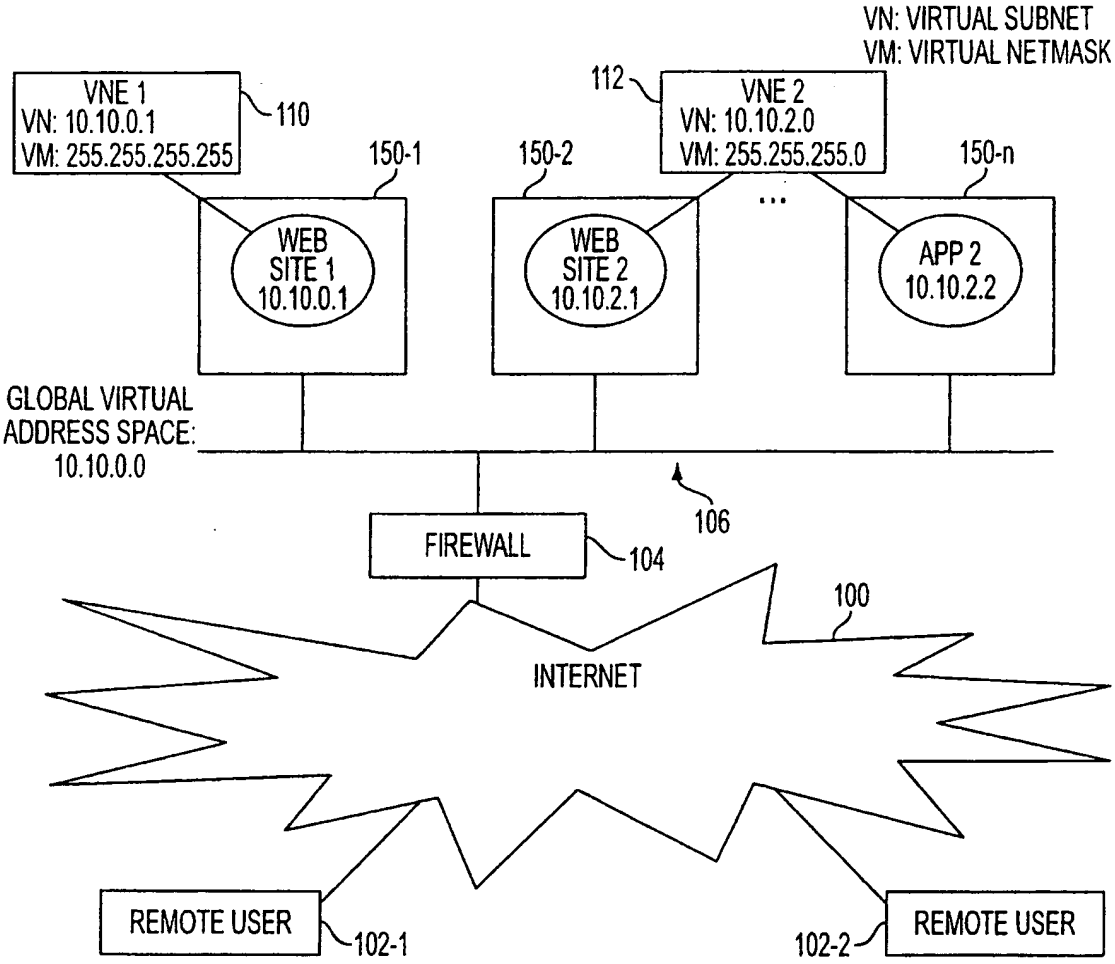

VN: VIRTUAL SUBNET
VM: VIRTUAL NETMASK
VNE 1
VN: 10.10.0.1
VM: 255.255.255.255
110
112
VNE 2
VN: 10.10.2.0
VM: 255.255.255.0
150-1
150-2
150-n
WEB SITE 1 10.10.0.1
WEB SITE 2 10.10.2.1
APP 2 10.10.2.2
GLOBAL VIRTUAL ADDRESS SPACE: 10.10.0.0
106
FIREWALL
104
100
INTERNET
REMOTE USER
102-1
102-2
REMOTE USER

FIG. 1

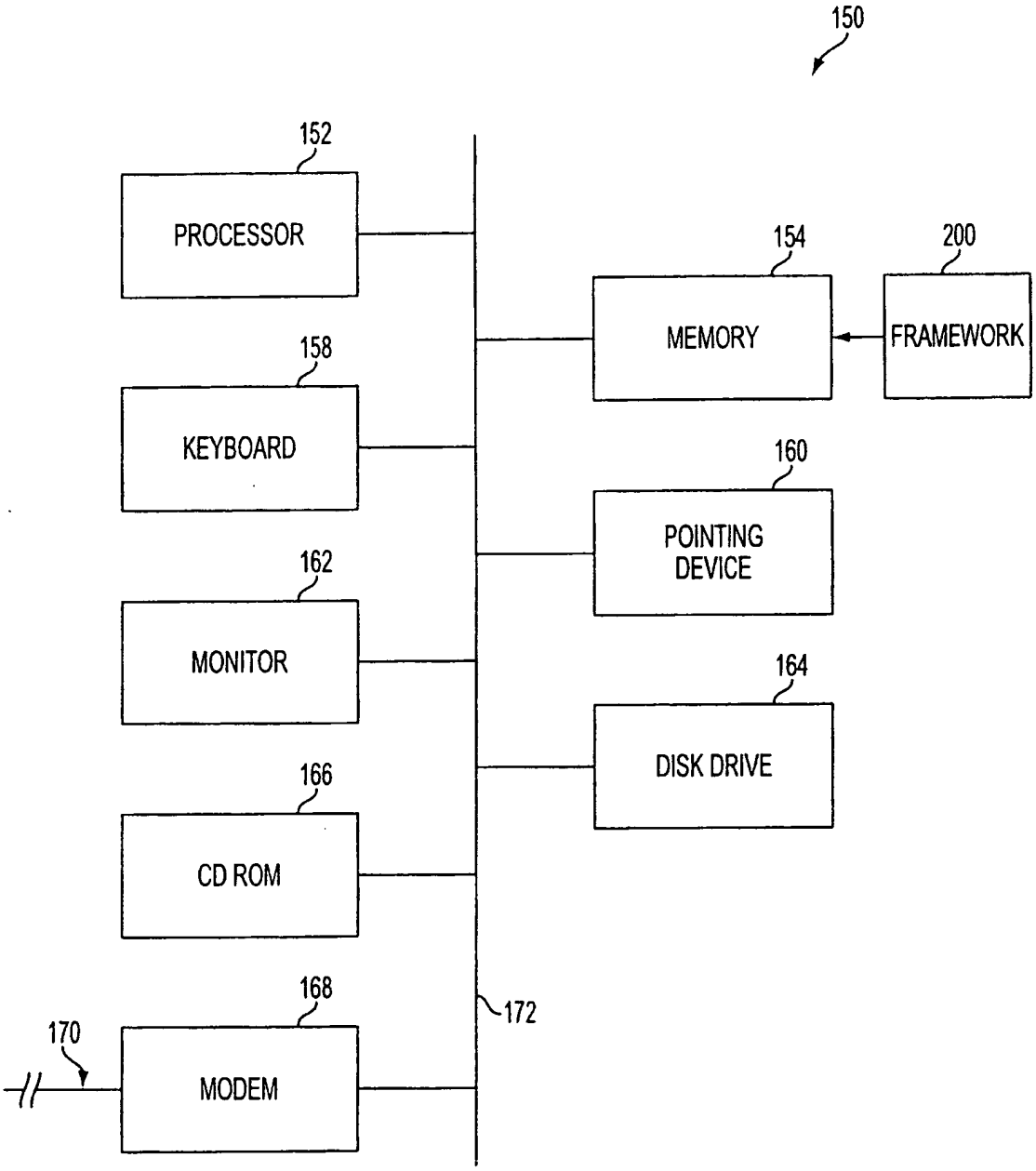

150
152
PROCESSOR
154
MEMORY
200
FRAMEWORK
158
KEYBOARD
160
POINTING DEVICE
162
MONITOR
164
DISK DRIVE
166
CD ROM
168
170
MODEM
172

FIG. 2

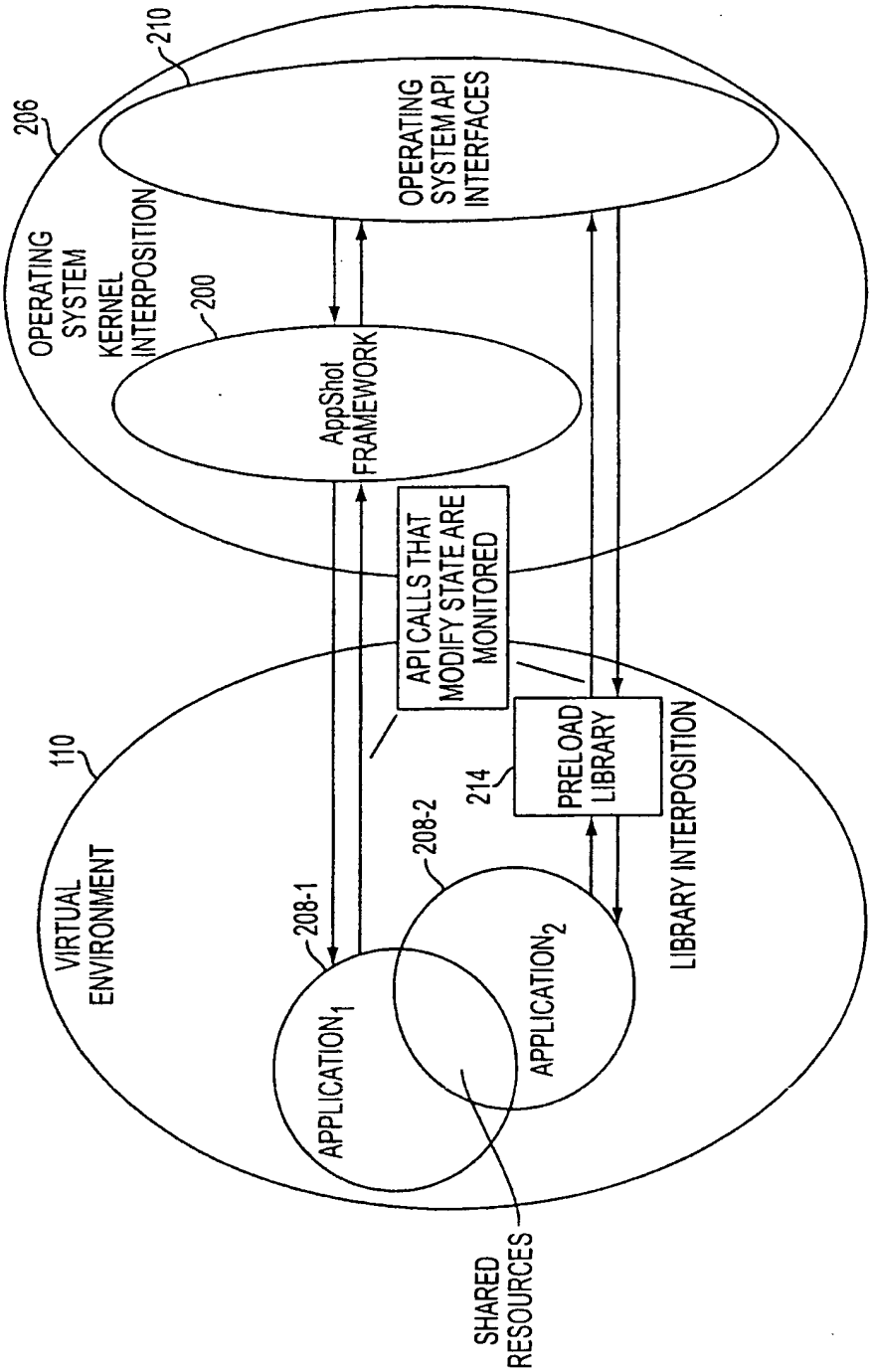

206
210
OPERATING SYSTEM API INTERFACES
OPERATING SYSTEM KERNEL INTERPOSITION
200
AppShot FRAMEWORK
API CALLS THAT MODIFY STATE ARE MONITORED
110
VIRTUAL ENVIRONMENT
214
PRELOAD LIBRARY
LIBRARY INTERPOSITION
208-2
208-1
APPLICATION2
APPLICATION1
SHARED RESOURCES

FIG. 3

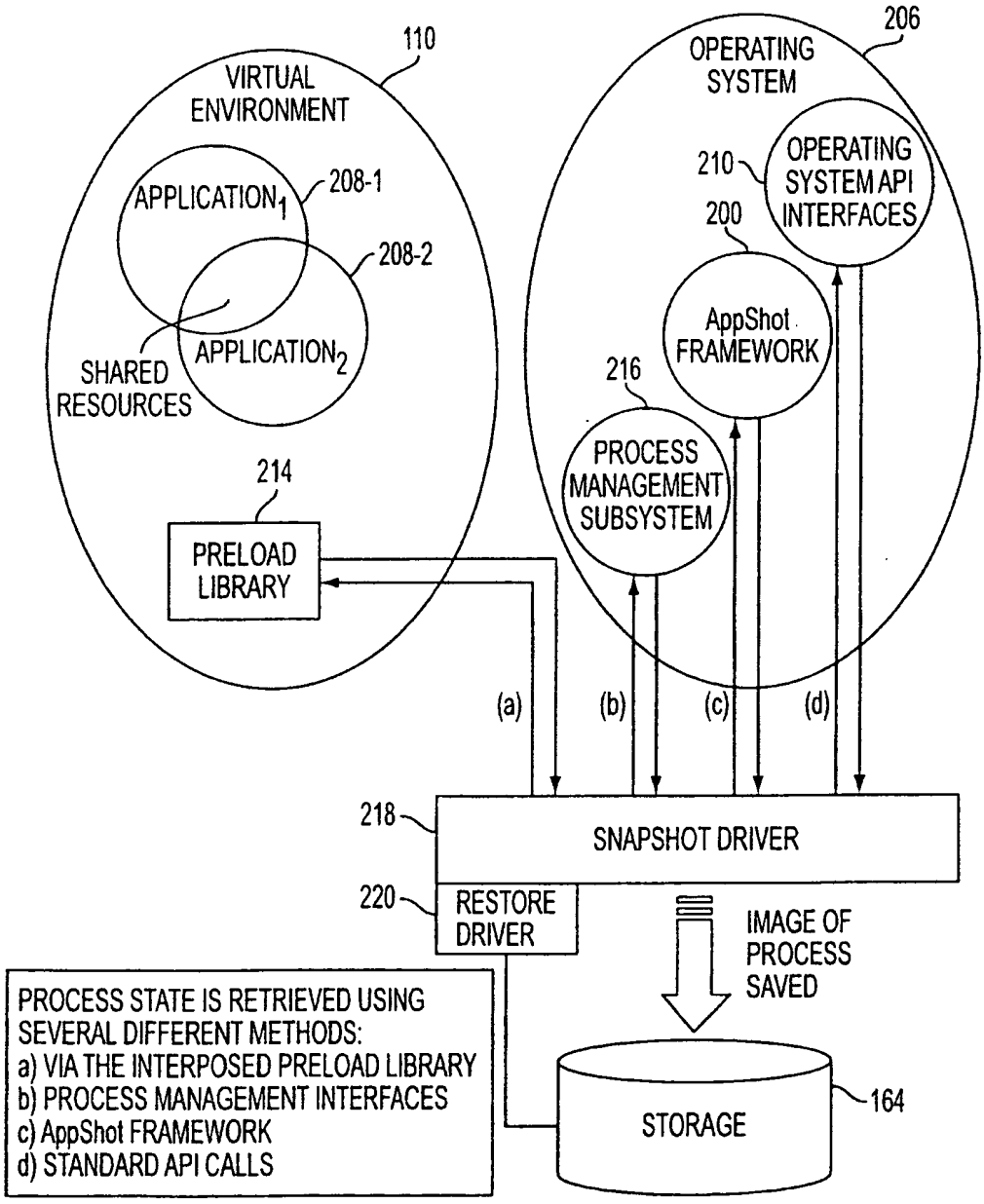
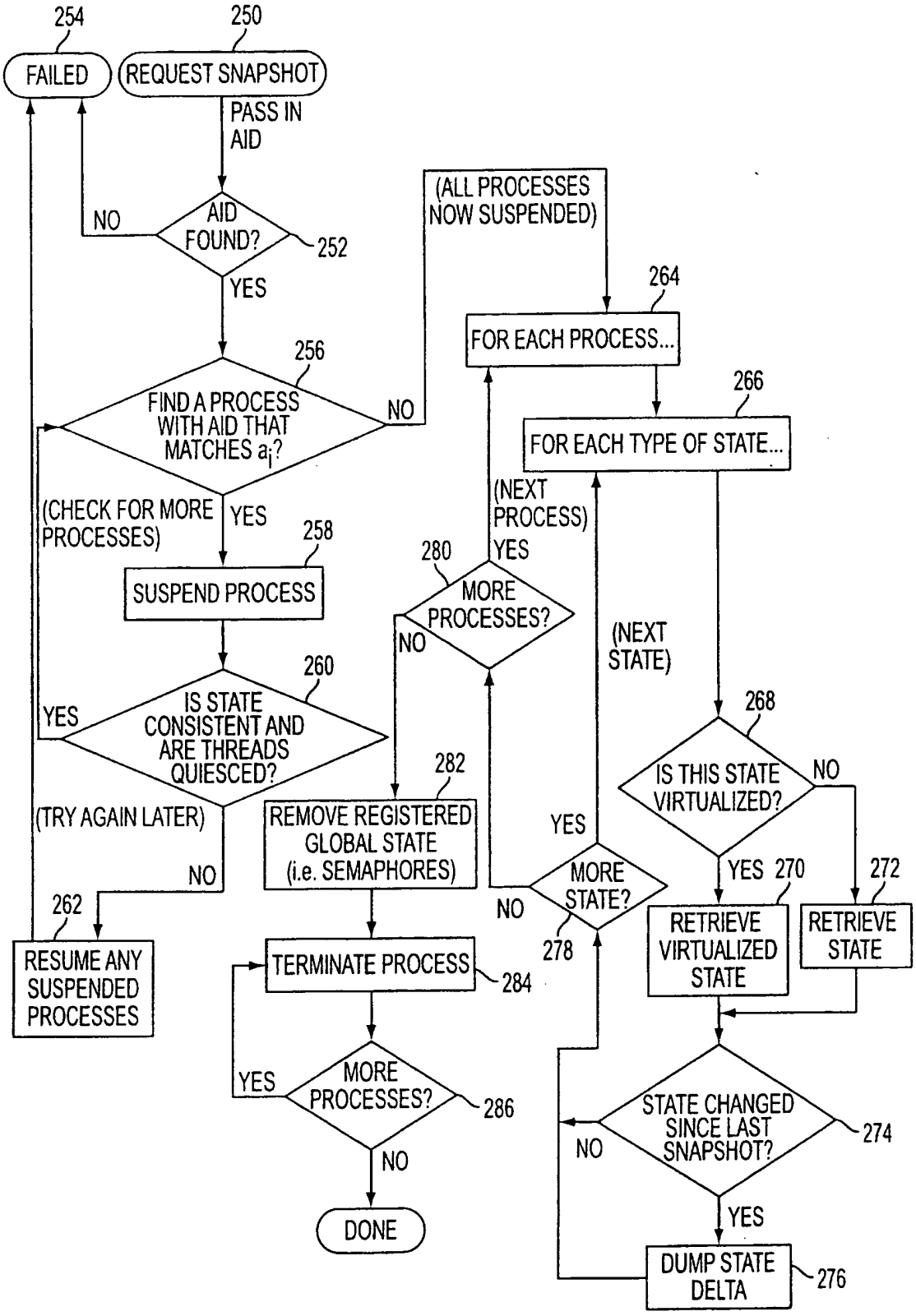

110
VIRTUAL ENVIRONMENT
APPLICATION1
208-1
208-2
APPLICATION2
SHARED RESOURCES
214
PRELOAD LIBRARY
206
OPERATING SYSTEM
210
OPERATING SYSTEM API INTERFACES
200
AppShot FRAMEWORK
216
PROCESS MANAGEMENT SUBSYSTEM
(a)
(b)
(c)
(d)
218
SNAPSHOT DRIVER
220
RESTORE DRIVER
IMAGE OF PROCESS SAVED
PROCESS STATE IS RETRIEVED USING SEVERAL DIFFERENT METHODS:
a) VIA THE INTERPOSED PRELOAD LIBRARY
b) PROCESS MANAGEMENT INTERFACES
c) AppShot FRAMEWORK
d) STANDARD API CALLS
STORAGE
164

FIG. 4


254
250
FAILED
REQUEST SNAPSHOT
PASS IN AID
NO
AID FOUND?
252
YES
256
FIND A PROCESS WITH AID THAT MATCHES a_i?
NO
(ALL PROCESSES NOW SUSPENDED)
(CHECK FOR MORE PROCESSES)
YES
258
SUSPEND PROCESS
260
YES
IS STATE CONSISTENT AND ARE THREADS QUIESCED?
(TRY AGAIN LATER)
NO
262
RESUME ANY SUSPENDED PROCESSES
264
FOR EACH PROCESS...
266
FOR EACH TYPE OF STATE...
(NEXT PROCESS)
280
YES
MORE PROCESSES?
NO
(NEXT STATE)
268
IS THIS STATE VIRTUALIZED?
NO
YES
270
272
RETRIEVE VIRTUALIZED STATE
RETRIEVE STATE
282
REMOVE REGISTERED GLOBAL STATE (i.e. SEMAPHORES)
YES
MORE STATE?
NO
278
TERMINATE PROCESS
284
YES
MORE PROCESSES?
286
NO
DONE
STATE CHANGED SINCE LAST SNAPSHOT?
274
NO
YES
DUMP STATE DELTA
276

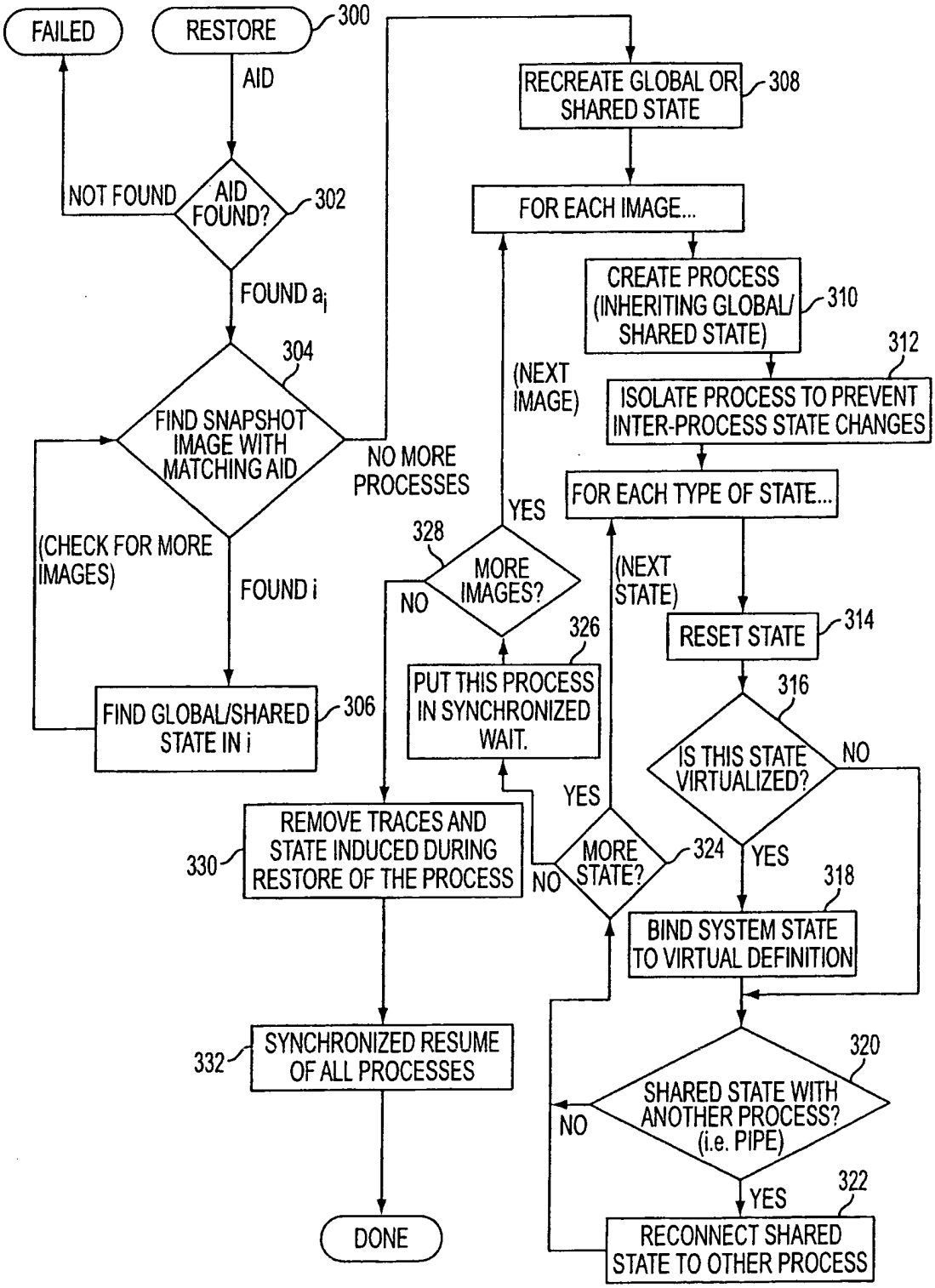

FAILED
RESTORE
300
AID
NOT FOUND
AID FOUND?
302
FOUND a_i
304
FIND SNAPSHOT IMAGE WITH MATCHING AID
NO MORE PROCESSES
(CHECK FOR MORE IMAGES)
FOUND i
FIND GLOBAL/SHARED STATE IN i
306
RECREATE GLOBAL OR SHARED STATE
308
FOR EACH IMAGE...
CREATE PROCESS (INHERITING GLOBAL/ SHARED STATE)
310
312
ISOLATE PROCESS TO PREVENT INTER-PROCESS STATE CHANGES
(NEXT IMAGE)
FOR EACH TYPE OF STATE...
YES
328
MORE IMAGES?
NO
(NEXT STATE)
RESET STATE
314
326
PUT THIS PROCESS IN SYNCHRONIZED WAIT.
316
IS THIS STATE VIRTUALIZED?
NO
YES
REMOVE TRACES AND STATE INDUCED DURING RESTORE OF THE PROCESS
330
MORE STATE?
324
NO
YES
318
BIND SYSTEM STATE TO VIRTUAL DEFINITION
320
SHARED STATE WITH ANOTHER PROCESS? (i.e. PIPE)
NO
YES
322
RECONNECT SHARED STATE TO OTHER PROCESS
SYNCHRONIZED RESUME OF ALL PROCESSES
332
DONE

FIG. 6

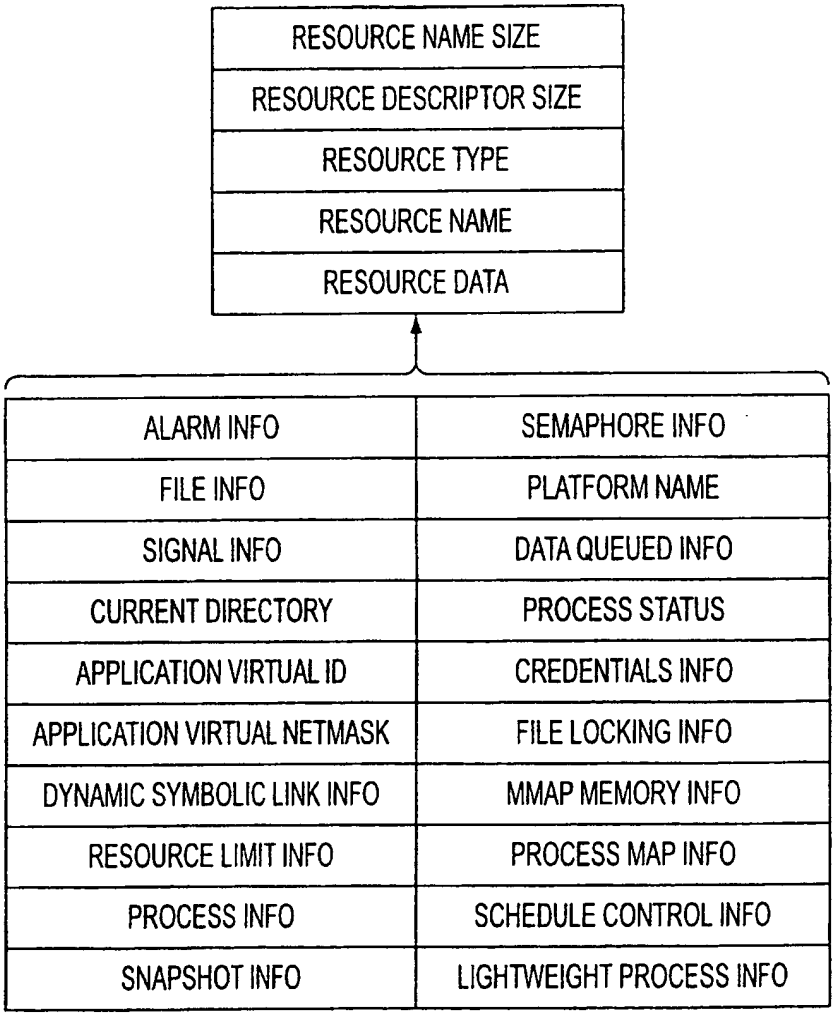

RESOURCE NAME SIZE
RESOURCE DESCRIPTOR SIZE
RESOURCE TYPE
RESOURCE NAME
RESOURCE DATA
ALARM INFO
SEMAPHORE INFO
FILE INFO
PLATFORM NAME
SIGNAL INFO
DATA QUEUED INFO
CURRENT DIRECTORY
PROCESS STATUS
APPLICATION VIRTUAL ID
CREDENTIALS INFO
APPLICATION VIRTUAL NETMASK
FILE LOCKING INFO
DYNAMIC SYMBOLIC LINK INFO
MMAP MEMORY INFO
RESOURCE LIMIT INFO
PROCESS MAP INFO
PROCESS INFO
SCHEDULE CONTROL INFO
SNAPSHOT INFO
LIGHTWEIGHT PROCESS INFO

FIG. 7

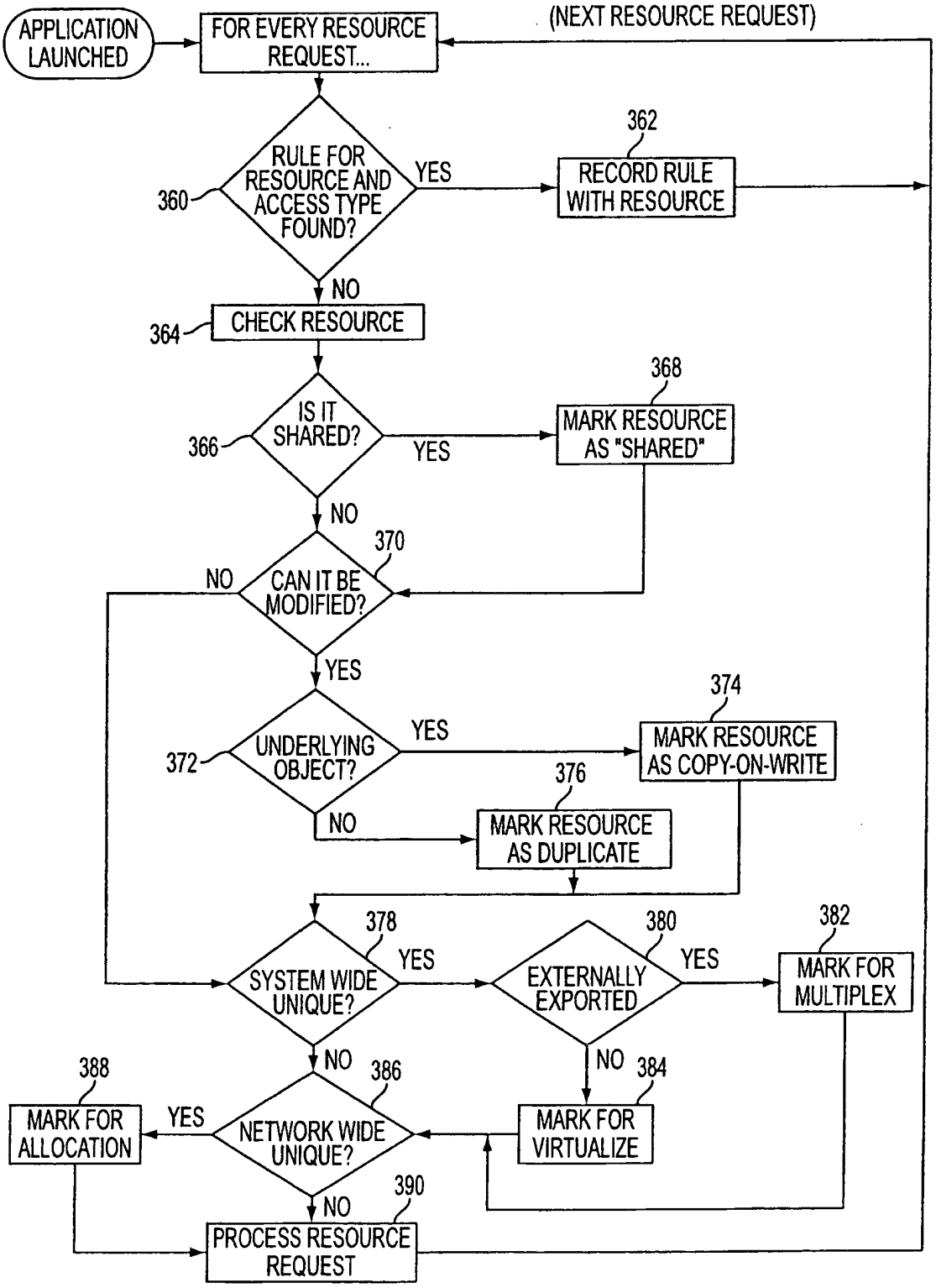

APPLICATION LAUNCHED
FOR EVERY RESOURCE REQUEST...
(NEXT RESOURCE REQUEST)
RULE FOR RESOURCE AND ACCESS TYPE FOUND?
360
YES
362
RECORD RULE WITH RESOURCE
NO
364
CHECK RESOURCE
IS IT SHARED?
366
YES
368
MARK RESOURCE AS "SHARED"
NO
CAN IT BE MODIFIED?
370
NO
YES
UNDERLYING OBJECT?
372
YES
374
MARK RESOURCE AS COPY-ON-WRITE
NO
376
MARK RESOURCE AS DUPLICATE
SYSTEM WIDE UNIQUE?
378
YES
EXTERNALLY EXPORTED
380
YES
382
MARK FOR MULTIPLEX
NO
NETWORK WIDE UNIQUE?
386
NO
384
MARK FOR VIRTUALIZE
388
MARK FOR ALLOCATION
YES
NO
390
PROCESS RESOURCE REQUEST

FIG. 8

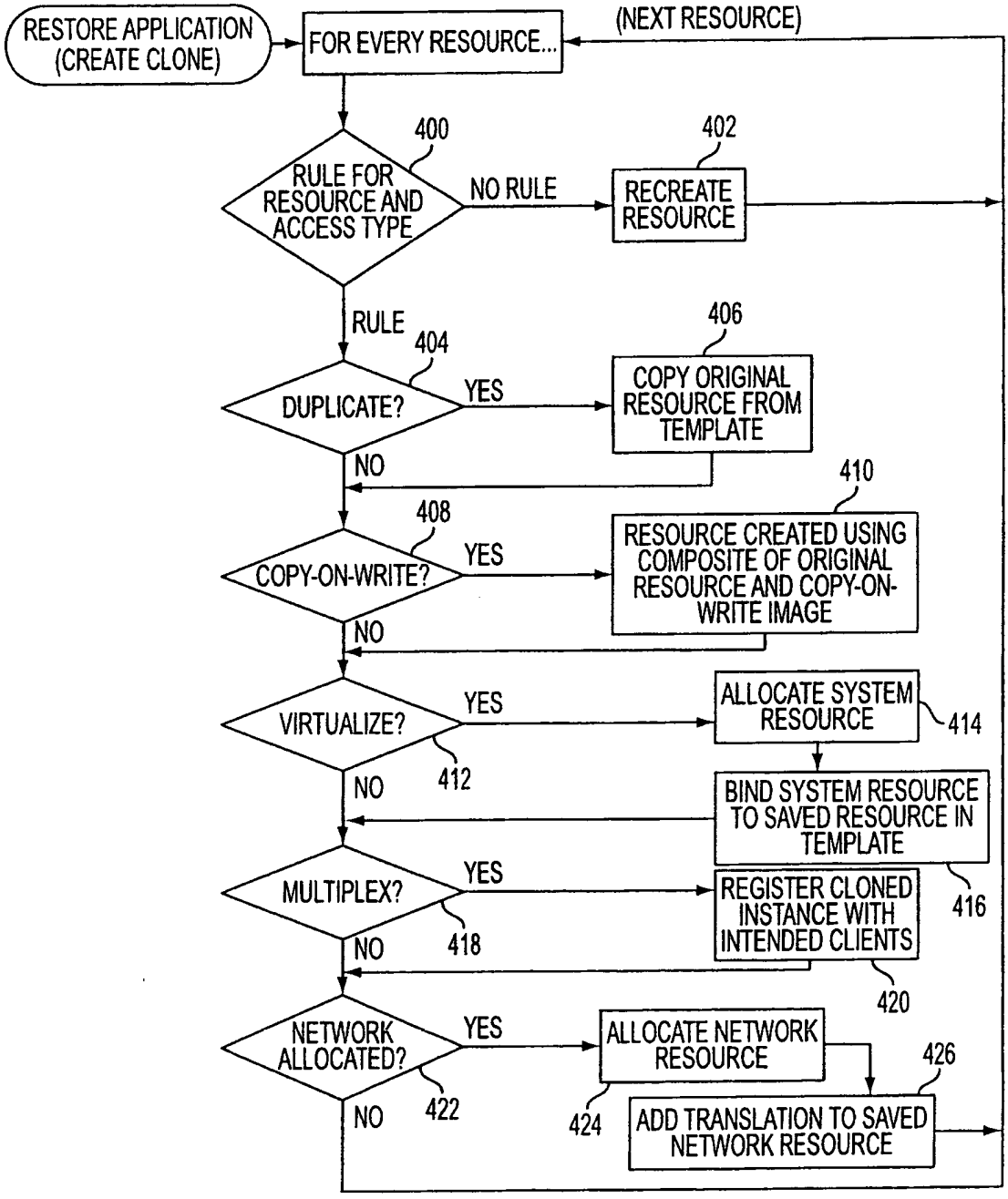

RESTORE APPLICATION (CREATE CLONE)
FOR EVERY RESOURCE...
(NEXT RESOURCE)
400
RULE FOR RESOURCE AND ACCESS TYPE
NO RULE
402
RECREATE RESOURCE
RULE
404
DUPLICATE?
YES
406
COPY ORIGINAL RESOURCE FROM TEMPLATE
NO
410
408
COPY-ON-WRITE?
YES
RESOURCE CREATED USING COMPOSITE OF ORIGINAL RESOURCE AND COPY-ON-WRITE IMAGE
NO
VIRTUALIZE?
YES
ALLOCATE SYSTEM RESOURCE
414
412
NO
BIND SYSTEM RESOURCE TO SAVED RESOURCE IN TEMPLATE
MULTIPLEX?
YES
REGISTER CLONED INSTANCE WITH INTENDED CLIENTS
416
418
NO
420
NETWORK ALLOCATED?
YES
ALLOCATE NETWORK RESOURCE
426
422
NO
424
ADD TRANSLATION TO SAVED NETWORK RESOURCE

FIG. 9

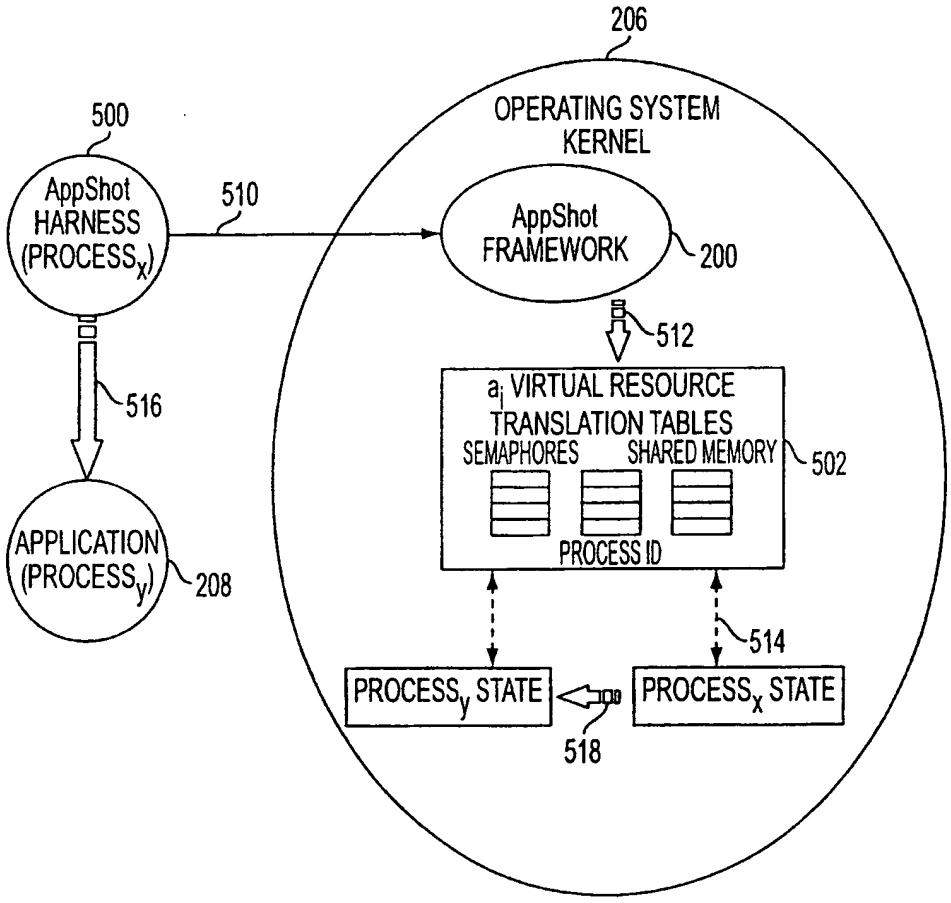

206
OPERATING SYSTEM KERNEL
500
AppShot HARNESS (PROCESS$_x$)
510
AppShot FRAMEWORK
200
512
a_i VIRTUAL RESOURCE TRANSLATION TABLES
SEMAPHORES
SHARED MEMORY
502
PROCESS ID
516
APPLICATION (PROCESS$_y$)
208
514
PROCESS$_y$ STATE
PROCESS$_x$ STATE
518

FIG. 10

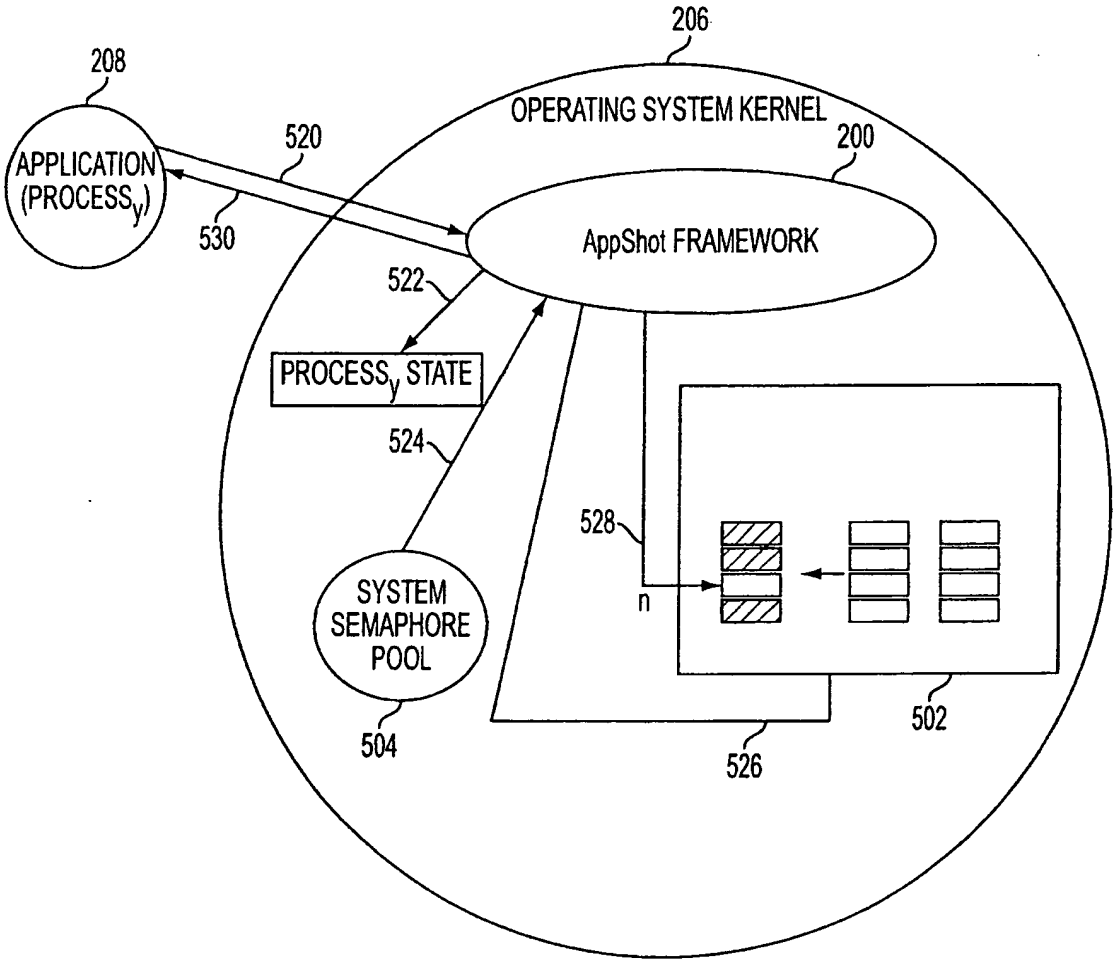

206
208
OPERATING SYSTEM KERNEL
200
APPLICATION
(PROCESS$_y$)
520
530
AppShot FRAMEWORK
522
PROCESS$_y$ STATE
524
528
n
SYSTEM
SEMAPHORE
POOL
504
502
526

FIG. 11

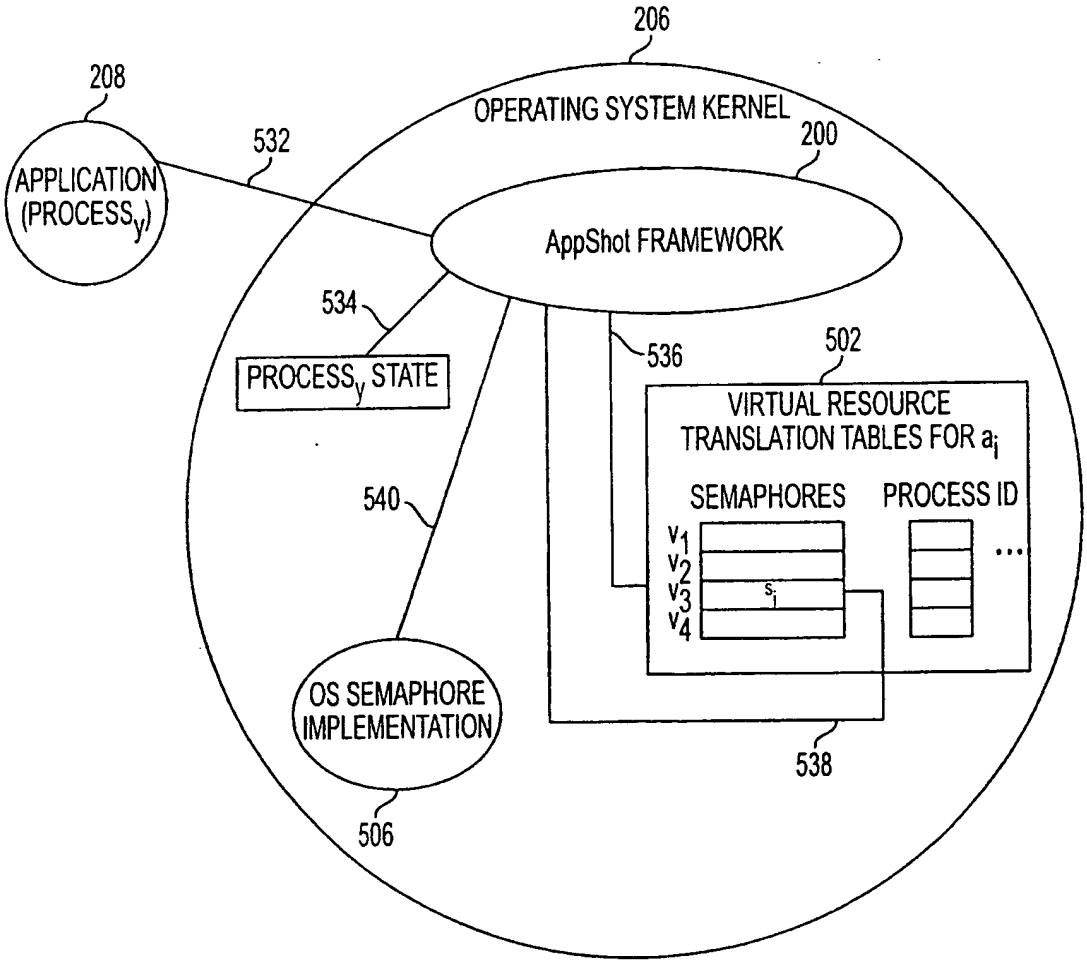

206
208
OPERATING SYSTEM KERNEL
532
200
APPLICATION (PROCESS$_y$)
AppShot FRAMEWORK
534
536
502
PROCESS$_y$ STATE
VIRTUAL RESOURCE TRANSLATION TABLES FOR a_i
SEMAPHORES
PROCESS ID
540
v_1
v_2
v_3
v_4
s_j
OS SEMAPHORE IMPLEMENTATION
538
506

FIG. 12

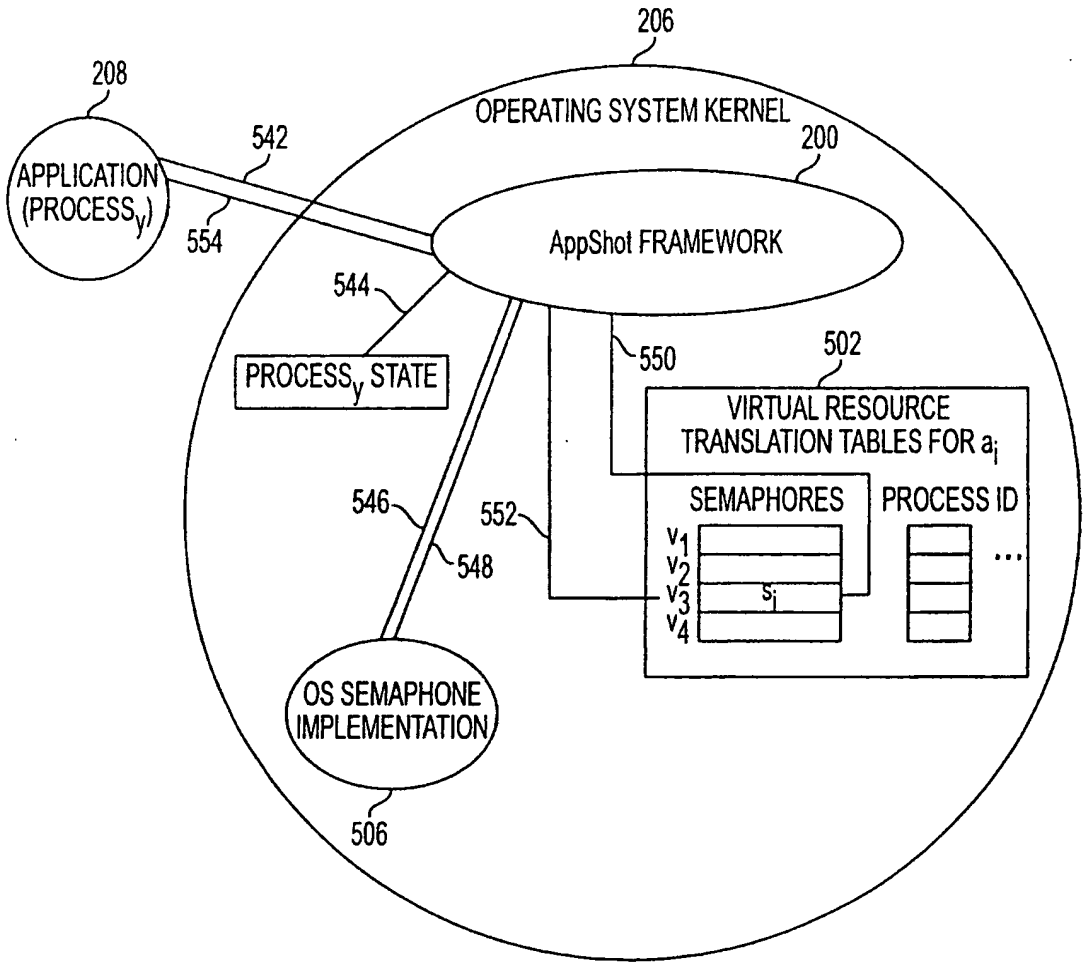

206
208
OPERATING SYSTEM KERNEL
200
542
APPLICATION
(PROCESS$_y$)
554
AppShot FRAMEWORK
544
PROCESS$_y$ STATE
550
502
VIRTUAL RESOURCE
TRANSLATION TABLES FOR a_i
SEMAPHORES
PROCESS ID
546
552
548
v_1
v_2
v_3
v_4
s_i
OS SEMAPHONE
IMPLEMENTATION
506

FIG. 13

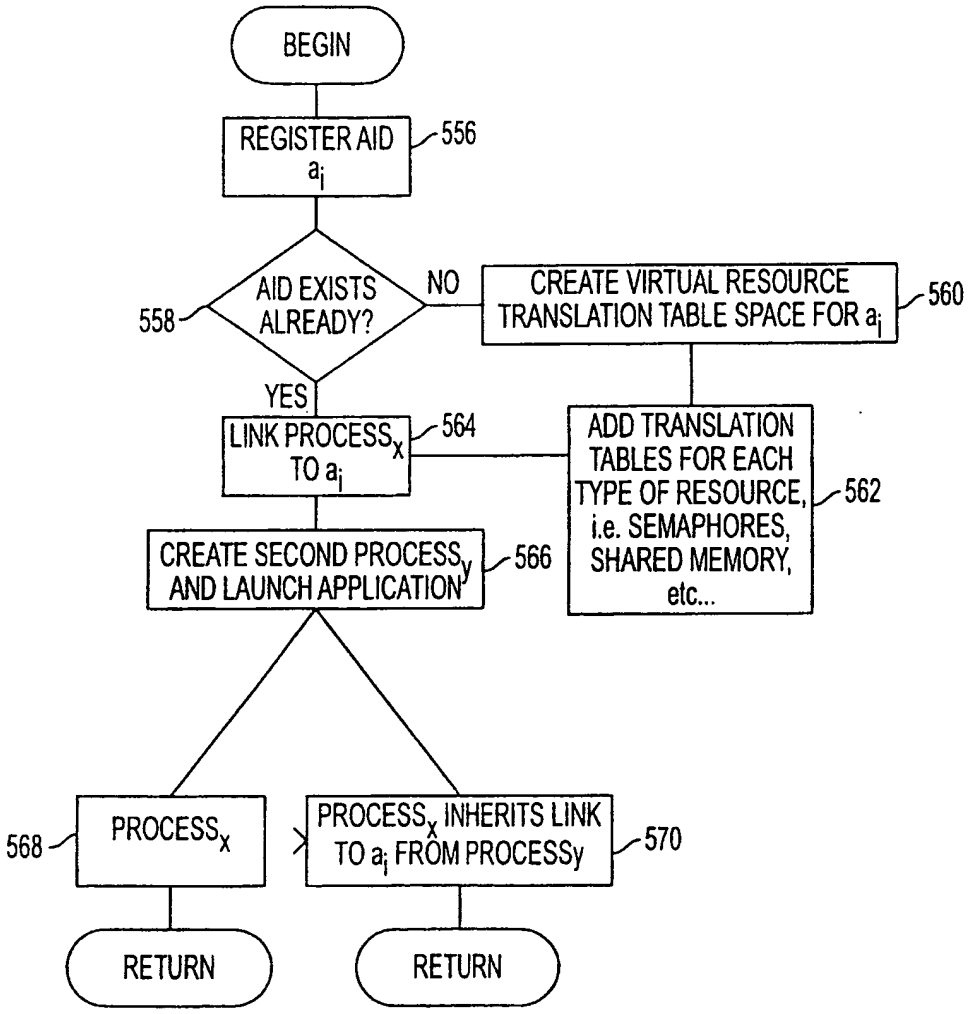

BEGIN
REGISTER AID a_i
556
AID EXISTS ALREADY?
558
NO
CREATE VIRTUAL RESOURCE TRANSLATION TABLE SPACE FOR a_i
560
YES
LINK PROCESS$_x$ TO a_i
564
ADD TRANSLATION TABLES FOR EACH TYPE OF RESOURCE, i.e. SEMAPHORES, SHARED MEMORY, etc...
562
CREATE SECOND PROCESS$_y$ AND LAUNCH APPLICATION
566
PROCESS$_x$
568
PROCESS$_x$ INHERITS LINK TO a_i FROM PROCESSy
570
RETURN
RETURN

FIG. 14

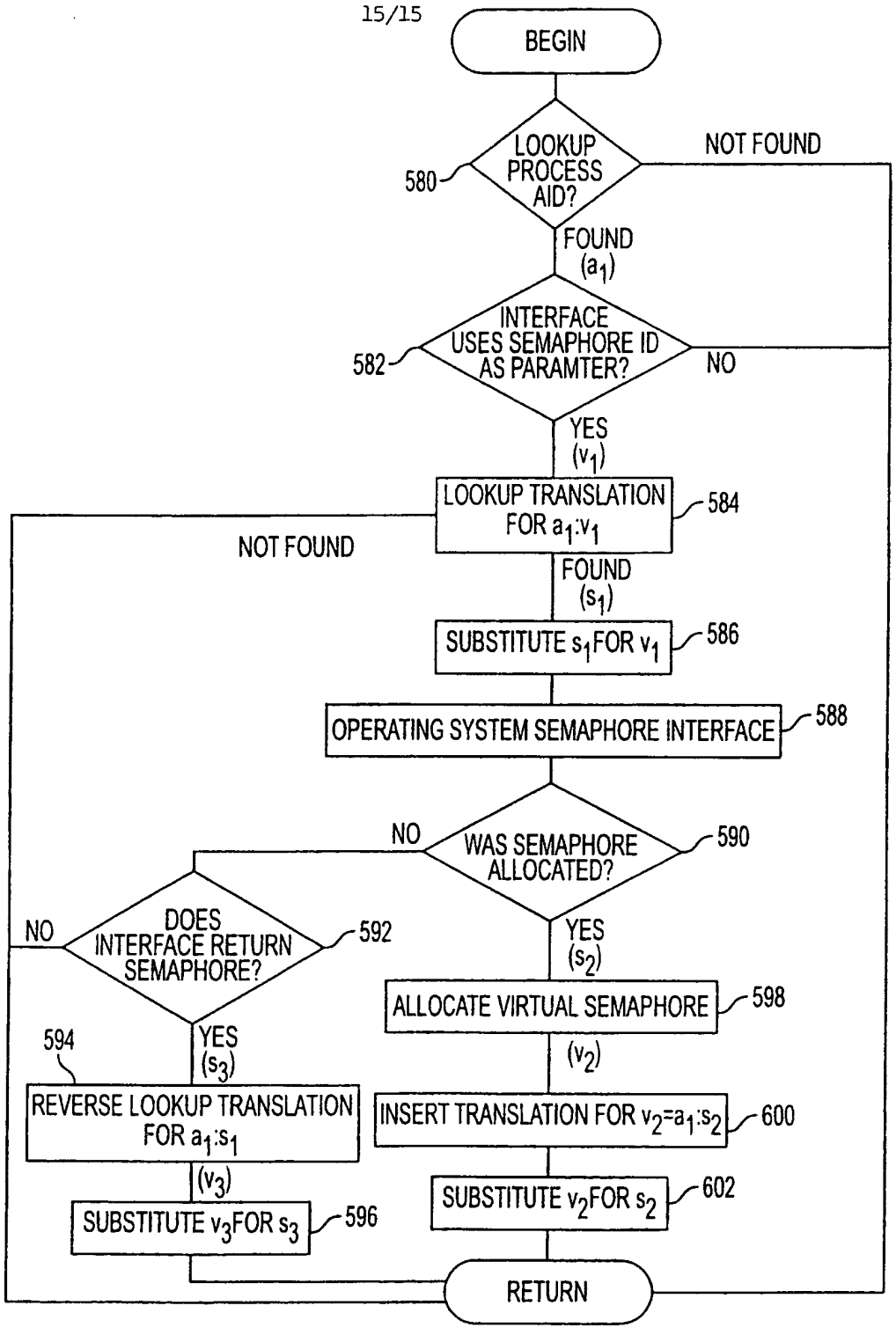

BEGIN
LOOKUP PROCESS AID?
580
NOT FOUND
FOUND (a_1)
INTERFACE USES SEMAPHORE ID AS PARAMTER?
582
NO
YES (v_1)
LOOKUP TRANSLATION FOR a_1:v_1
584
NOT FOUND
FOUND (s_1)
SUBSTITUTE s_1FOR v_1
586
OPERATING SYSTEM SEMAPHORE INTERFACE
588
NO
WAS SEMAPHORE ALLOCATED?
590
NO
DOES INTERFACE RETURN SEMAPHORE?
592
YES (s_2)
ALLOCATE VIRTUAL SEMAPHORE
598
(v_2)
INSERT TRANSLATION FOR v_2=a_1:s_2
600
SUBSTITUTE v_2FOR s_2
602
YES (s_3)
594
REVERSE LOOKUP TRANSLATION FOR a_1:s_1
(v_3)
SUBSTITUTE v_3FOR s_3
596
RETURN

FIG. 15

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/27640

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 3/14
US CL :709/204; 345/330
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 709/204; 345/330

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

IEEE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,530,795 A (WAN) 25 June 1996, col. 2, line 30 - col. 3, line 23.	1-6
Y	US 5,822,523 A (ROTHSCHILD et al) 13 October 1998, col. 16, lines 22-35.	1-6

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 18 DECEMBER 2000	Date of mailing of the international search report 09 JAN 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer MARK RINEHART James R. Matthews Telephone No. 703-305-9600

Form PCT/ISA/210 (second sheet) (July 1998)*